The C++/CX String^ is not an object, even though it wears a hat

devblogs.microsoft.com/oldnewthing/20211230-00

December 30, 2021

Raymond Chen

C++/CX refers to Windows Runtime strings as `Platform::String^` with a hat instead of a star. But even though the string wears a hat, it is not an `Object^`.

The `String^` type is a representation of the Windows Runtime `HSTRING`. And of the rules of `HSTRING` is that a null pointer is a valid `HSTRING`, and it represents the empty string, that is, a string with no characters.

This makes `String^` a strange sort of beast.

```
String^ s = L""; // sets s = nullptr
```

If you assign an empty string to it, you get `nullptr` back.

```
void f(String ^s)
{
    if (s) { /* string is not empty */ }
}
```

Testing a `String^` against `nullptr` tests whether the string is empty.

```
String ^s = nullptr; // represents empty string
auto data = s->Data(); // legal! returns pointer to L""
auto length = s->Length(); // legal! returns 0.
auto equal = s->Equals(L"nope"); // legal! returns false.
```

That's right: I dereferenced a null pointer *and it felt good*.

Calling methods on a null `String^` pointer is legal, and the operations are performed on an empty string.

This weird behavior of null `String^` pointers has consequences beyond just strings. If you convert a null `String^` to `Object^` (a boxing operation), the null-ness is *preserved*:

```
String^ s = L""; // s is nullptr
Object^ o = s; // o is nullptr!
```

This differs from the behavior in other projections like C#, JavaScript, and C++/WinRT, where boxing an empty string produces a non-null object (that in turn holds an empty string).

The fact that a `String^` is not an `Object^` means that you cannot reinterpret between them.

```
String^ s = /* some value */;
Object^ o = reinterpret_cast<Object^>(s); // crash
```

The `reinterpret_cast` will treat a `String^` as an `Object^`. But a `String^` is secretly a `HSTRING`, whereas an `Object^` is secretly an `IInspectable*`. The reinterpret-cast tells the compiler to treat this `HSTRING` as if it were an `IInspectable*`, and bad things happen, since the compiler is going to try to call the `AddRef` method from the `IInspectable`'s vtable, but `HSTRING`s don't have a vtable, much less a vtable with `AddRef` in slot 1.

What you need to do is box the string into an object and unbox the object back into a string.

```
Object^o = s; // box the string into an object
String^s = static_cast<String^>(o); // unbox the object into a string
```

Bonus chatter: C++/CX delegates are also not objects, even though they too wear hats.

Raymond Chen

Follow